Exploring the impact of situational context – A case study of a software development process for a microservices architecture

Rory V. O'Connor
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
+353-1-700-5643
Rory.OConnor@dcu.ie

Peter Elger
NearForm Ltd.
Suite 420 Mountain View
CA 94040, USA
+1-916-235-6459
Peter.Elger@nearform.com

Paul M. Clarke
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
+353-1-700-7021
Paul.M.Clarke@dcu.ie

ABSTRACT
Over the decades, a variety of software development processes have been proposed, each with their own advantages and disadvantages. It is however widely accepted that there is no single process that is perfectly suited to all settings, thus a software process should be molded to the needs of its situational context. In previous work, we have consolidated a substantial body of related research into an initial reference framework of the situational factors affecting the software development process. Practitioners can consult this framework in order to profile their context, a step necessary for effective software process decision making. In this paper, we report on the findings from a case study involving process discovery in a small but successful and growing software development firm. In this organization, which has a focus on continuous software evolution and delivery, we also applied the situational factors reference framework, finding that context is a complex and key informant for software process decisions. Studies of this type highlight the role of situational context in software process definition and evolution, and they raise awareness not just of the importance of situational context, but also of the complexity surrounding software process contexts, a complexity which may not be fully appreciated in all software development settings.

CCS Concepts
• Software and its engineering → Software creation and management → Software development process management → Software development methods.

Keywords
Software Development Process; Software Development Context; Agile; Lean; Process Selection.

1. INTRODUCTION
Given the proliferation of software development models, methods and standards that have been proposed over the years, it is not surprising to discover that there has also been much debate regarding the effectiveness of various software development approaches. It is generally accepted that no single software development process is perfectly suited to all software development settings [1] and no setting is unchanging [2]. Therefore some amount of process adaption and situational tailoring [3] is required in order to render a process suitable to a given situational context. As has been noted in the literature a software process is a continuous rather than a static concern [4] and so we should seek to identify techniques that can improve our understanding of interactions between software processes and their situational contexts [5]. Accordingly an optimal software development process can be regarded as being dependent on the situational characteristics of individual software development settings. Such characteristics include the nature of the application(s) under development, team size, requirements volatility and personnel experience

In certain quarters of the present software development business environment, continually changing situational contexts are fueling the customer demand for rapid evolution of software products. Now more than even in the history of the software production business, software development organizations are under enormous pressure to evolve software intensive systems through the release of valuable software in increasingly shorter time durations. Whereas at one stage software releases would occur one or two times per year, now given current competitive market opportunities this has been reduced to weekly, daily and even hourly time periods. Organizations therefore need to innovate and release software in faster parallel cycles of days or even hours, and this has involved the adoption of certain new practices in industry. In this paper, we present the results from a case study in one such organization, where a continuous software evolution and delivery model has been implemented and evolved to meet the demands of the situational context. This study shows that situational context, whilst being a complex concept, is a key informant for software process selection and design.

This paper is organized as follows: Section 2 outlines the situational factors framework; Section 3 presents an overview of the company studied, including its software development process; Section 4 examines the role of situational context; and finally, Section 5 presents a discussion and conclusion.

2. SITUATIONAL FACTORS
The importance of context in software process decisions has been acknowledged for some time [6]. Whilst the literature has noted that "the organization's processes operate in a business context that should be understood" [7] and that a "life cycle model… [should be] appropriate for the project's scope, magnitude, complexity, changing needs and opportunities" [8], contributions

ICCSP'16, May 14-16, 2016, Austin, TX, USA.

DOI: http://dx.doi.org/10.1145/12345.67890

to the literature in relation to software process context space are lacking. Software development necessarily occurs in a development context, which includes a large number of concerns and factors [9, 10] and it is this contextualization which provides a better understanding of what works for whom, where, when, and why [11]. In support of the importance of understanding the impact of situational factors, authors such as Dyba [12] point out that it is this dependence on a potentially large number of context variables in any study that is an important reason for why software engineering is so hard.

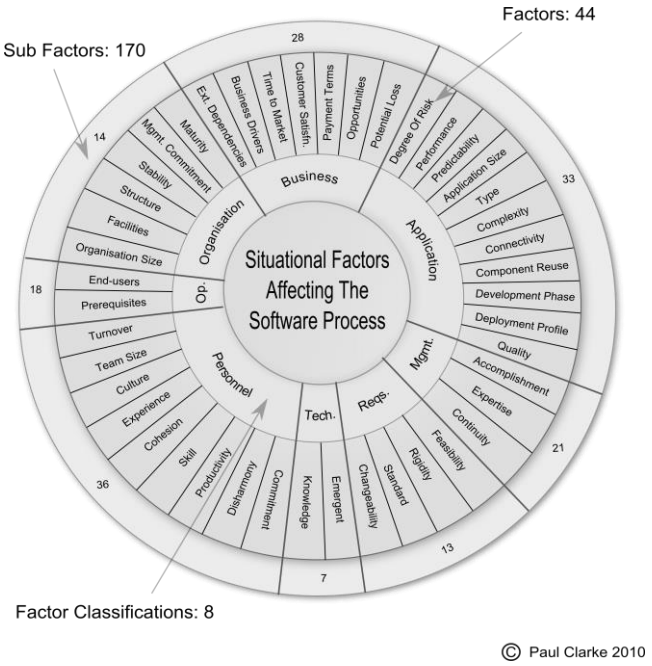

Figure 1. Situational Factors Reference Framework

Despite the frequent references to the importance of situational context in the literature, it was the apparent lack of a comprehensive situational factors framework for software development that led two of the authors to produce and publish an initial reference framework [5], itself an amalgamation of earlier contributions, from multiple areas such as risks, estimation, etc.

Table 1. Situational Factors Classification

Classification	Description
Personnel	Constitution and characteristics of the non-managerial personnel involved in the software development efforts.
Requirements	Characteristics of the requirements.
Application	Characteristics of the application(s) under development.
Technology	Profile of the technology being used for the software development effort.
Organization	Profile of the organization.
Operation	Operational considerations and constraints.
Management	Constitution and characteristics of the development management team.
Business	Strategic and tactical business considerations.

The framework incorporates 44 individual factors (ref. Figure 1) which are categorized using 8 classifications (ref. Table 1), and which are based upon 170 underlying sub-factors. A sample listing of the sub-factors in the *Personnel* classification is presented in Table 2.

Table 2. Personnel Factors & Sub-Factors

Factor	Sub-Factor
Turnover	Turnover of personnel
Team size	(Relative) team size
Culture	Team culture/resistance to change
Experience	General team experience / diversity/ ability to understand the human implications of a new information system/team ability to work with management/application experience/analyst experience/programmer experience/tester experience/experience with development methodology / platform experience.
Cohesion	General cohesion/team members who have not worked for you/team not having worked together in the past/team ability to successfully complete a task/team ability to work with undefined elements and uncertain objectives / overdependence on team members / distributed team/ team geographically distant.
Skill	Operational knowledge/team expertise (task) / team ability to work with undefined elements and uncertain objectives/training development.
Productivity	Team ability to carry out tasks quickly / general productivity.
Commitment	Commitment to project among team members.
Disharmony	Interpersonal conflicts.
Changeability	Scope creep/continually changing system requirements/ill-defined project goals / gold plating/unclear system requirements.

The situational factors reference framework is in the view of its authors a stepping stone towards greater appreciation of the complexity of software development settings, and the rigorous approach employed in its creation from a rich variety of sources has given rise to a framework that they consider to present a broadly informed reference for the software development community [13]. Using the framework, the situational factors affecting the software process were examined in practice as part of a case study, details of which are presented in the following sections.

3. CASE STUDY COMPANY

The case study firm NearForm Ltd., is a software development company with a presence in the US and Europe and which has experienced substantial growth through the continual delivery of high quality software to some of the largest companies in the world, including blue chip financial institutions. Value is a key focus in the NearForm lifecycle and it is concerned with an acute responsiveness to client needs (be they new features or defect resolutions). The organization works to a regular 5-day iteration for software development, deploying working software weekly(sometimes daily) through a standard feature bundle. While regular iterations can be predictable from the outset, continual analysis of the value stream ensures that each iteration may be re-planned in real time, delivering the highest possible value from organizational capacity (ref. Figure 2).

Whilst it is acknowledged that tooling can affect the design of a software process [14], the impact of technology on shaping the

process in this case is profound and may even run contrary to the Agile Manifesto value of '*Individuals and interactions over processes and tools*'. Within NearForm the continual software evolution and delivery is made possible through the aggressive incorporation of contemporary and predominately open source software tools. While the speedy delivery of innovative features is a vital enabler of competitive advantage, it is only effective if it is accompanied by reliable and high quality deployments.

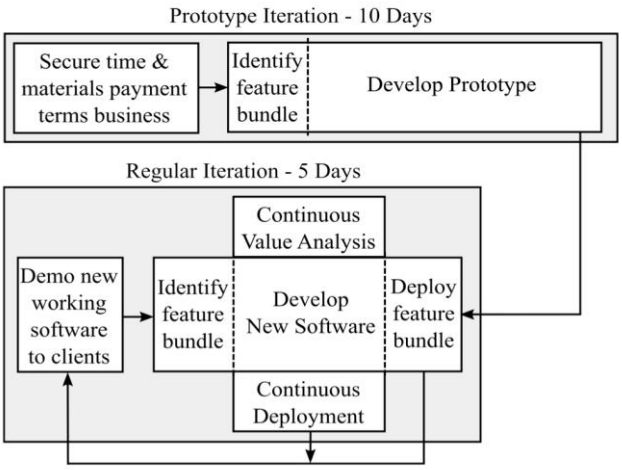

Figure 2. NearForm Process Lifecycle

There are four key technologies driving the process architecture: (1) Java-script and Node.js which enable extremely rapid code development by utilizing the same programming language across the entirety of the system; (2) Alongside a distributed micro-services architecture, under which the system is broken down into a set of discreet co-operating processes, typically each service is of the order of several hundred lines of code only; (3) This architectural approach is coupled with a continuous deployment model, layered over the Docker container engine, whereby individual services (or several services at a time) may be deployed without perturbing the system as a whole; (4) Finally the company ensures quality through steps such as code commit hooks via GitHub (for distributed revision control and source code management) and the Travis Continuous Integration tool set. Together, these technologies enable the company to perform well under a time and materials contract basis, whereby clients are initially attracted through the rapid delivery of a prototype in 10 days, and thereafter, regular iterations of new working software are reviewed every 5 days.

3.1 Java-script and Node.js

Once considered a 'toy' language by many developers [15], Java-script now presents as an ideal language for full-stack, enterprise development [16]. Node.js when coupled with its supporting package management system – *npm* - provides a lean and efficient platform that enables developers to be highly productive. This, when combined with an effective front-end framework (such as *angular* or *react*) provides a powerful and rapid development platform enabling the same language to be used in all tiers. The rapid adoption of node.js is evidenced by Figure 3, which shows the number of open source modules available for the various popular open source platforms (Node.js is the top line). As of January 2016, there are over 225,000 modules available for node.js with module downloads running in excess of 2.5 billion per month [17], a very strong indicator that this technology stack has some significant momentum behind it.

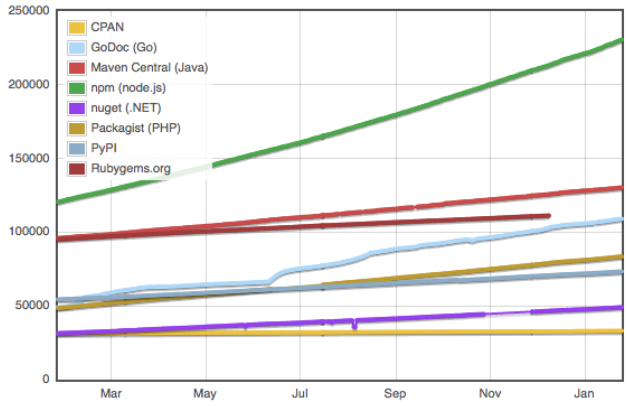

Figure 3 module counts

3.2 Micro-service Architecture

The term micro-service architecture refers to a style of development under which a system is broken down into a number of small co-operating components [18]. Typically these components interact over a direct point-to-point interface (for example, http). As with all architectural styles, there are pros and cons to micro-services. Key benefits include: a highly modular and decoupled system that can be easier to maintain than a traditional class hierarchy; the ability to deploy services rapidly to a production system – because services are independent entities, only the service under question need undergo rigorous testing and the rest of the system has not been changed; finally, micro-services are highly cohesive units of code that are easier to reason about and manage in isolation, this tends to reduce the burden on developers and if implemented responsibly can lead to simpler code with less defects.

As a corollary to these benefits, micro-service systems require a more sophisticated DevOps infrastructure [19], typically requiring the construction of a service deployment pipeline. Use of cloud and container technologies enables the construction of such pipelines and it is this technology enabler that is driving the adoption of these hyper-agile, lean processes. It is the final piece in the jigsaw that makes the technology stack so powerful.

3.3 Software Container Technology

Software containers provide a means of encapsulating functionality within an isolated process space, i.e. a single operating system level process can attend to just a specific, small piece of executable code. The concept of software containers originated in the late seventies with the addition of the *chroot* system call to the BSD Unix operating system. This feature was largely unused until FreeBSD *jails* were introduced in 2000. This was followed by Solaris *zones* in 2004. A more mainstream user-land implementation in the Linux kernel followed in 2008 with the advent of *LXC-Containers*. However the technology first began to gain wide adoption in 2013 via the Docker project, and it has resulted in the capability of developers to regularly inject new, easily digestible features into live systems with less risk than traditional software development and deployment models.

Container technology may become the mainstream for certain types of software development, especially with the development of container management and orchestration systems such as *Kubernetes*, *Docker Swarm* and *AWS container services*.

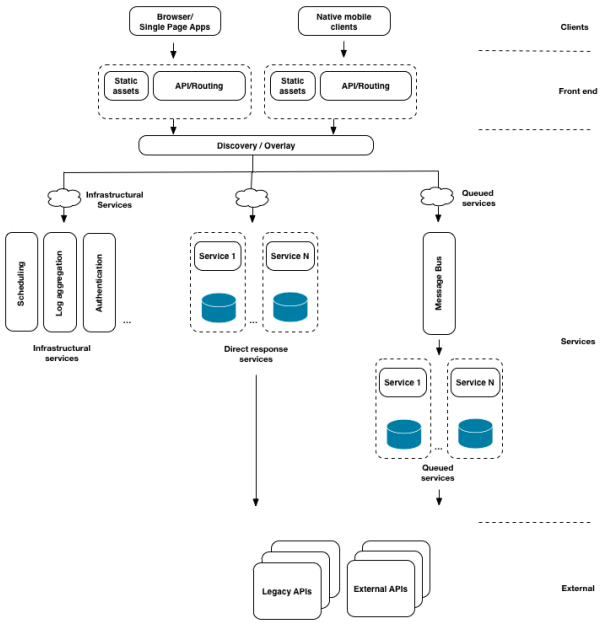

Figure 4 micro-service reference architecture

4. APPLYING THE SITUATIONAL FACTORS REFERENCE FRAMEWORK

Two researchers in association with the Director of Engineering from NearForm undertook a detailed analysis of the company's situational factors, the primary results of which are presented in Table 3.

Table 3. Situational Factors Identified in Case Study

Factors Identified in Case Study	
Personnel	**Cohesion**: The company has a geographically distributed team which whose effectiveness is made possible through the adoption of tools, especially with respect to geographically diverse programming as supported by GitHub; **Culture**: The team culture has a low resistance to change, change is in fact promoted as a highly desirable characteristic and it is enabled at a technical level through the various tools and technologies identified in this paper; **Experience, Skill & Productivity**: The experience, skill and productivity of personnel are all at the upper end of the scale – what are sometimes referred to as premium people. The staff cohort in the company tend to be of high to very high core technical competency, with the result that individuals may operate fluidly and efficiently without the need for extensive training or up-skilling; **Turnover**: Personnel turnover is low (especially with key technical staff) with the result that continuity of technical excellence and know-how is high, there is therefore a reduced need for documented artefacts in relation to product architecture and process descriptions.
Requireme	**Changeability:** Requirements are subject to frequent, sudden and significant change, a reality of operating in a fast moving and highly innovative market. As a result, a lean/agile approach to software development (such as was outlined in Section 3) is preferable for this setting.
Application	**Quality:** Operational product quality requirement is high and the technology adopted, including Continuous Integration systems, assist greatly in achieving product quality targets; **Type:** The applications under development and evolution (though requiring a high level of quality) do not need to be at the level of safety-critical software, nor are they directly affected by market regulation. As a result, a lean process, enabled via the technology and development stack, is suitable for the needs of this organization.
Technology	**Emergent:** The technology is emergent and innovative thus there is a high level of adoption of new technologies and tools to enable process initiatives. Embracing the rapid supporting technology offerings means that the process itself is subject to change as a result of technology strengths and limitations. This too is a feature of the context that has reduced the desirability of precise and extensive process descriptions which would continually need to be revisited as a result of the rapid pace of change.
Org.	**Size:** Organizational size is small – with the result that information exchange and communications can occur efficiently through video conferences or calls or face-to-face meetings thus enabling more agile/lean software approaches.
Operation	**End-Users:** Operational end-users of the software are open to changing requirements and rapidly evolving software systems. In fact, end-users are in this case demanding such capability from their software supplier in pursuit of competitive advantages in a fast moving market. This fact is key in shaping much of the process design – which is capable of working to a time and materials payments model and accommodates rapidly changing requirements.
Management	**Expertise & Accomplishment:** Management expertise and accomplishment is high in key markets and product technology stacks, meaning that the business can pivot in harmony with the emerging technology without the risk of the business and technical strategic directions becoming discommoded.
Business	**Time to Market:** The company are in a fast moving market where the need for rapid delivery is paramount (smooth, regular and rapid delivery is enabled through the adoption of a microservices architecture along with deployment infrastructure such as Docker); **Business Drivers:** The company's business drivers are leveraged upon vanguard activities in key open source emerging technologies - technical excellence and high levels of innovation are key to differentiation and business development; **Payment Arrangements:** Payment terms tend to be time and materials based which supports the type of near-real feature elaboration with clients that is made possible by the micro services architecture.

5. DISCUSSION

There is no one size or style that fits all when it comes to software development processes. The process form and content is determined by a complex cocktail of situational circumstances that may well be unique to each development team, with the circumstances themselves being in constant flux. The general domain of situational factors affecting the software development

process may be viewed as being strategically important to the future of software development. It is the authors' view that efforts to reveal the nature of the interrelationship between a process and its context should be encouraged, even if it is a complex undertaking that should be approached with care. The case study reported upon in this paper represents one small step towards a robust understanding of the interplay between a process and its context, while also highlighting the continuum that is the software process concern - since the NearForm process that was discovered as part of this research and which is described in this paper, a contemporary real-world effective software process, would barely have been imaginable to earlier generations of software developers. And the authors suggest that it is emerging developments in technology and tooling that are perhaps the primary reason that the process identified in this paper is even possible; an observation that may be incongruent with the Agile Manifesto value of '*Individuals and interactions over processes and tools*'.

Our case study also serves to demonstrate not just the relationship between certain situational factors and software process decisions, it also offers evidence of the complexity of the interplay between a process and its context. Although our research is still on going and there are limitations and threats to validity (which cant be expressed here for space reasons) it is already clear that no less than 17 individual situational factors are key informants of the software development process in the case of the company under examination. These factors touch on every category of situational context, ranging from basic business factors, to technology factors, to application and product factors, to organizational considerations, to requirements characteristics, and also to operational end-user demands. These are broad concerns, which must all be satisfied by an appropriate process.

Software process decisions are therefore multi-layered and complex, perhaps more so than may be appreciated in all quarters. And this complex and fluid software process decision chain which interacts with its context may account for the absence of a generalised software process approach that is perfectly suited to all settings – quite simply because the vast diversity of software development contexts beguiles and undermines attempts to develop a universally applicable process model.

6. REFERENCES

[1] P. Clarke, R. O'Connor, B. Leavy and M. Yilmaz. "Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance," IEEE Transactions on Software Engineering, vol. 41, no. 12, pp. 1169-1183, 2015.

[2] R. V. O'Connor and P. Clarke. "Software process reflexivity and business performance: Initial results from an empirical study," Proceedings of the 2015 International Conference on Software and System Process, pp. 142-146, 2015.

[3] G. Coleman and R. O'Connor. "Investigating software process in practice: A grounded theory perspective," Journal of Systems and Software, vol. 81, no. 5, pp. 772-784, 2008.

[4] B. Curtis. "Three problems overcome with behavioral models of the software development process," 11th International Conference on Software Engineering, pp. 398-399, 1989.

[5] P. Clarke and R. V. O'Connor. "The situational factors that affect the software development process: Towards a comprehensive reference framework," Journal of Information and Software Technology, vol. 54, no. 5, pp. 433-447, 2012.

[6] P. Feiler and W. Humphrey, Software Process Development and Enactment: Concepts and Definitions. CMU/SEI-92-TR-004. Pittsburgh, Pennsylvania, USA: Software Engineering Institute, Carnegie Mellon University, 1992

[7] SEI, CMMI for Development, Version 1.3. CMU/SEI-2006-TR-008. Pittsburgh, PA, USA: Software Engineering Institute, 2010.

[8] P. Clarke, R.V. O'Connor and M. Yilmaz. "A hierarchy of SPI activities for software SMEs: results from ISO/IEC 12207-based SPI assessments." In Software Process Improvement and Capability Determination, pp. 62-74. Springer Berlin Heidelberg, 2012.

[9] L. McLeod, and S. G. MacDonell. "Factors that affect software systems development project outcomes: A survey of research." ACM Computing Surveys (CSUR) 43, no. 4, 2011.

[10] W. Orlikowski. "CASE tools as organizational change: Investigating incremental and radical changes in systems development." MIS quarterly, pp. 309-340, 1993.

[11] T. Dyba. "Contextualizing empirical evidence." Software, IEEE 30, no. 1, 81-83, 2013.

[12] T Dyba, D. Sjiberg and D. Cruzes, "What works for whom, where, when, and why? On the role of context in empirical software engineering," in Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on , vol., no., pp.19-28, 20-21 Sept. 2012.

[13] P. Clarke and R. V. O'Connor. "Changing Situational Contexts Present a Constant Challenge to Software Developers." In Systems, Software and Services Process Improvement, pp. 100-111. Springer International Publishing, 2015.

[14] M. Mora, O. Gelman, R. O'Connor, F. Alvarez, and J. Macías-Lúevano. "A Conceptual Descriptive-Comparative Study of Models and Standards of Processes in SE, SwE, and IT Disciplines Using the Theory of Systems." Emerging Systems Approaches in Information Technologies: Concepts, Theories, and Applications: Concepts, Theories, and Applications, 2009.

[15] H.M. Kienle. "It's about time to take JavaScript (more) seriously." IEEE Software, 27, no. pp. 60-62, 2010.

[16] A. Nitze. "Evaluation of JavaScript Quality Issues and Solutions for Enterprise Application Development." In Software Quality. Software and Systems Quality in Distributed and Mobile Environments, pp. 108-119. Springer International Publishing, 2015.

[17] E. DeBill, "Comparison of how many packages are available across different programming languages", 2016. http://www.modulecounts.com/.

[18] J. Thones. "Microservices". IEEE Software, 32, no. 1, pp.116-116, 2015.

[19] L. Bass, I. Weber, and L. Zhu. "DevOps: A Software Architect's Perspective". Addison-Wesley Professional, 2015.